Licenciatura Engenharia Informática e Multimédia

Infraestruturas Computacionais Distribuídas

Semestre de Verão 2022 / 2023

Trabalho Prático 2

Docente Diogo Remédios

18 de Julho de 2023

Trabalho realizado por:

Fábio Dias, nº 42921

Índice

Índice de Figuras

1. Introdução

Para a segunda fase do trabalho foi-nos pedido para desenvolver o jogo “Quatro-em-Linha” para o browser e ser possível a realização de jogos entre jogadores que possuam o Cliente, estrutura previamente desenvolvida, e pelo browser.

Previamente foram criadas duas simples estruturas: Servidor e Cliente. Este Servidor vai ser necessário dado que é o intermediário entre a realização dos jogos, tanto de Cliente contra Cliente, como Cliente contra browser e ainda a possibilidade de browser contra browser.

Para isto será necessário o desenvolvimento de um Servidor Web assim como diversas estruturas, nomeadamente as páginas HTML que serão apresentadas no browser, o seu CSS para customizar visualmente a página, os ficheiros javascript para oferecer dinamismo à página. Ainda são necessárias as Servlets para processar os pedidos por parte do browser.

2. Servidor Web

Um Servidor Web é uma aplicação ligada à internet que é responsável por atender pedidos de clientes. Também possui um repositório de recursos necessários para o funcionamento da mesma.

O Servidor Web não possui estado, ou seja, não guarda qualquer informação acerca das interações previamente efetuadas entre este e o Cliente. Para obter informações acerca do Cliente, foi desenvolvido um parâmetro que é enviado nos cabeçalhos dos pedidos feitos pelos Cliente denominado de "*cookie*".

3. Desenvolvimento

Como mencionado, foram desenvolvidas páginas HTML para serem apresentadas no browser para o cliente. Estas foram efetuadas com recurso às JSP's (JavaServer Pages) onde é possível desenvolver páginas HTML com auxílio a código Java. Existem três JSP's neste Servidor Web: "index.jsp", "playerLobby.jsp" e "gameWindow.jsp".

Também são necessários dois ficheiros javascript, um para o "playerLobby.jsp" e outro para o "gameWindow.jsp".

Por fim, é essencial para o bom funcionamento da aplicação, as Servlets. Estas permitem estender a funcionalidade de um Servidor Web. São classes que agem como mini servidores, aumentado a sua capacidade fornecendo funcionalidades adicionais. São persistentes, simples, flexíveis, eficientes e portáveis.

Para esta parte são necessárias sete Servlets: "SignInLoginController", "GetInitialInfo", "GetUpdateInfo", "AnswerChallenge", "GameWindowInitialInfo", "GameWIndowUpdateGame" e "PlayColumn". A primeira está associada ao "index.jsp", as três seguintes ao "playerLobby.jsp" e as últimas três ao "gameWindow.jsp".

4. Fluxo de Utilização

O fluxo de execução é bastante simples: ao aderir ao Servidor Web, o browser recebe a página index.jsp. Introduz os dados e realiza o "SignIn" ou o "Login". Este pedido é enviado para a Servlet "SignInLoginController" que por sua vez verifica os dados recebidos e desenvolve uma resposta afirmativa ou negativa. Caso seja positiva, o browser será redirecionado para a "playerLobby.jsp", caso seja negativa, será reenviado para o "index.jsp" para efetuar a ação novamente.

No "playerLobby.jsp" é imediatamente apresentado os dados do cliente, assim como os jogadores já presentes no Servidor. A cada segundo, pelo javascript, é efetuado um pedido para a Servlet "GetUpdateInfo" que retorna, caso exista, a mensagem mais antiga que o cliente recebeu. Caso seja necessário atualizar a página, assim é feito, dinamicamente. Isto serve para a remoção ou adição de jogadores na lista e desafios efetuados. Quando o cliente é desafiado, um menu aparece à frente da lista que informa qual o jogador que o desafiou. Neste menu, é possível aceitar ou rejeitar o desafio. Caso rejeite, a janela simplesmente fecha; caso aceite, o browser é redirecionado para a janela "gameWindow.jsp".

Aqui é apresentado o nome do adversário do jogador, assim como a tabela do Quatro em Linha. A cada segundo é efetuada uma atualização, via javascript, à Servlet "GameWindowUpdateGame" que, caso possua uma mensagem, vai atualizar a tabela e proporcionar os botões ao jogador. Efetuando as jogadas, a tabela vai sendo atualizada até, eventualmente, existir um vencedor, perdedor ou haver um empate. Aqui, um menu irá aparecer que reencaminha o jogador de volta à "playerLobby.jsp".

5. Desenvolvimento da Aplicação

No "index.jsp" foi usado um *"form"* para submeter os dados para a Servlet "SignInLoginController". Esta envia uma mensagem para o Servidor, a partir da classe Cliente criada na primeira parte do trabalho prático. Este interpreta os dados e envia uma resposta. Esta pode possuir diferentes tipos de erros, mas se tudo correr bem, é efetuado o reencaminhamento do cliente para o "playerLobby".

```
<body>
<form action="SignInLoginController" method="post">
    <div class="container">
        <label for="playerName">Name: </label>
        <input class="textField" id="playerName" name="playerName" type="text"/>
        <label for="playerPassword">Password: </label>
        <input class="textField" id="playerPassword" name="playerPassword" type="text"/>
        <br>
        <br>
        <input name="Input" class="signInLogin" type="submit" value="SignIn">
        <input name="Input" class="signInLogin" type="submit" value="Login">
    </div>
    <div>
    </div>
</form>

</body>
```

Figura 1 - index.jsp

```
HttpSession session = request.getSession();

String playerName = request.getParameter("playerName");
String playerPassword = request.getParameter("playerPassword");
```

Figura 2 – SignInLoginControlller – Obter parametros

```
//Open Connection to Game Server
Client client = null;

if(session.getAttribute("client") == null)
{
    client = new Client("localhost", 50000);
    session.setAttribute("client", client);
}
else
{
    client = ((Client) session.getAttribute("client"));
    //client.clearMessages();
}
```

Figura 3 - SignInLoginControlller – Cliente

```
if(received != null)
{
    //System.out.println("RECEIVED SIGNIN" + received);
    if(Message.analiseMessage(received)[1].equals("SignIn-PlayerAlreadyExists") ||
       Message.analiseMessage(received)[1].equals("Login-PlayerDoesNotExist") ||
       Message.analiseMessage(received)[1].equals("Login-PasswordMismatch"))
    {
        response.sendRedirect("/"+getServletContext().getServletContextName() +"/index.jsp");
    }
    else
    {
        String[] parameters = Message.analiseMessage(received);
        session.setAttribute("playersInfo", Arrays.copyOfRange(parameters, 9, parameters.length));
        response.sendRedirect("/"+getServletContext().getServletContextName() +"/playerLobby.jsp");
    }
}
else
{
    response.sendRedirect("/"+getServletContext().getServletContextName() +"/index.jsp");
}
```

Figura 4 - SignInLoginControlller – Reencaminhament

Ao chegar ao "playerLobby.jsp" são efetuadas duas ações via javascript. A primeira é obter a informação que a mensagem de resposta do SignInLogin enviou. Esta foi armazenada nos dados de sessão como parâmetro na Servlet acima referida. Para obter estes dados, que agora sim são importantes, é necessário dar uso à Servlet "GetInitialInfo". A partir dos métodos do javascript, a informação dinâmica é adicionada à página HTML.

```
<div class="playerLobbyContainer">
  <div class="playerLobbyTitle">
    <h1>Player Lobby</h1>
  </div>

  <div class="content">
    <div class="playerInfo">
        <p id="playerInfoName">Test</p>
            <div class="playerInfoLabel">
            <label>Games Played: </label>
            <span id="playerInfoGamesPlayed">2</span>
            </div>
        <div class="playerInfoLabel">
            <label>Games Won: </label>
            <span id="playerInfoGamesWon">1</span>
        </div>
        <div class="playerInfoLabel">
            <label>Games Lost: </label>
            <span id="playerInfoGamesLost">1</span>
        </div>
    </div>

    <div id="playersList">
    </div>
```

Figura 5 - playerLobby.jsp

```javascript
window.onload = function()
{
    setInitialInfo();
    setInterval(updateInfo, 1000);
}
```

Figura 6 - Javascript - playerLobby.jsp – window.onload

```javascript
function setInitialInfo()
{
    xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function(){
        if(xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            res = xmlhttp.responseText;
            var initialInfo = res.split(" ");

            var playerInfoName = document.getElementById("playerInfoName");
            var playerInfoGamesPlayed = document.getElementById("playerInfoGamesPlayed");
            var playerInfoGamesWon = document.getElementById("playerInfoGamesWon");
            var playerInfoGamesLost = document.getElementById("playerInfoGamesLost");

            playerInfoName.innerHTML = initialInfo[0];
            playerInfoGamesPlayed.innerHTML = initialInfo[1];
            playerInfoGamesWon.innerHTML = initialInfo[2];
            playerInfoGamesLost.innerHTML = initialInfo[3];

            for(var playerIndex = 4; playerIndex < initialInfo.length; playerIndex = playerIndex + 4)
            {
                createPlayerEntry(initialInfo[playerIndex],
                                  initialInfo[playerIndex + 1],
                                  initialInfo[playerIndex + 2],
                                  initialInfo[playerIndex + 3])
            }
        }
    }
    xmlhttp.open("GET","GetInitialInfo", true);
    xmlhttp.send();
}
```

Figura 7 - Javascript - playerLobby.jsp – setInitialInfo

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    response.setContentType("application/json");
    ServletOutputStream out = response.getOutputStream();

    HttpSession session = request.getSession();

    ClientRegistry myInfo = ((ClientRegistry) ((Client) session.getAttribute("client")).getMyInfo());
    String[] playersInfo = ((String[]) session.getAttribute("playersInfo"));

    //My Info
    out.print(myInfo.getClientName() + " " +
              Integer.toString(myInfo.getGamesPlayed()) + " " +
              Integer.toString(myInfo.getGamesWon()) + " " +
              Integer.toString(myInfo.getGamesLost()));

    //Players Info
    for(int playerIndex = 0; playerIndex < playersInfo.length; playerIndex = playerIndex + 4)
    {
        out.print(" " + playersInfo[playerIndex] + " " + playersInfo[playerIndex + 1] + " " + playersInfo[playerIndex + 2] + " " + playersInfo[playerIndex + 3]);
    }

    out.println();
}
```

Figura 8 - Servlet – GetInitialInfo

A outra ação definida no “window.onload” do javascript é um temporizador que chama o método “updateInfo” a cada segundo. Este por sua vez faz um pedido à Servlet “GetUpdateInfo” que devolve os parâmetros para possíveis mensagens cujo cliente tenha recebido. Este método, dependendo do tipo de mensagem que recebe, vai executar o código apropriado.

```
function updateInfo()
{
    xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function(){
        if(xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            res = xmlhttp.responseText;
            var updatedInfo = res.split(" ");

            switch(updatedInfo[0])
            {
                case "AddPlayer":

                    createPlayerEntry(updatedInfo[1], updatedInfo[2], updatedInfo[3], updatedInfo[4]);

                    break;

                case "RemovePlayer":

                    removePlayerEntry(updatedInfo[1]);

                    break;

                case "Challenged":

                    console.log("Entrei");
                    challengedByPlayer(updatedInfo[1]);

                    break;

                default:

                    break;
            }
        }
    }
    xmlhttp.open("GET","GetUpdateInfo", true);
    xmlhttp.send();
}
```

Figura 9 - Javascript - playerLobby.jsp – updateInfo

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    response.setContentType("application/json");
    ServletOutputStream out = response.getOutputStream();

    HttpSession session = request.getSession();

    Client client = ((Client) session.getAttribute("client"));

    if(client != null)
    {
        String message = client.getLastReceivedMessage();

        if(message != null)
        {
            String[] parameters = Message.analiseMessage(message);
            for(int parameter = 0; parameter < parameters.length; parameter++)
            {
                out.print(parameters[parameter]);

                if(parameter < parameters.length - 1)
                {
                    out.print(" ");
                }
            }

            out.println();
        }
    }
}
```

Figura 10 - Servlet – GetUpdateInfo

Uma das possíveis mensagens que o jogador pode receber é uma mensagem de desafio. Isto é, um jogador está a desafiá-lo para um jogo de Quatro em Linha. Como já foi mencionado, um menu aparece à frente da lista dos jogadores e fornece duas opções ao cliente: “Aceitar” ou “Rejeitar”.

“Rejeitar” simplesmente remove essa janela enquanto “Aceitar” chama um método no javascript que faz um pedido à Servlet “AnswerChallenge” que simplesmente redireciona o browser para a “gameWindow.jsp”.

```javascript
function answerChallenge(answer)
{
    document.getElementById("challengeAnswerParam").setAttribute("value", answer);

    xmlhttp=new XMLHttpRequest();
    xmlhttp.open("GET","AnswerChallenge", false);
    xmlhttp.send();
}
```

Figura 11 - Javascript - playerLobby.jsp – answerChallenge

```java
String startedTheGame = request.getParameter("startedTheGame");
System.out.println(startedTheGame);

if(startedTheGame != null)
{
    System.out.println("Entered the Game IF");
    session.setAttribute("startedTheGame" , startedTheGame);
}

if(challengeAnswer.equals("true"))
{
    System.out.println("Entered redirect");

    response.sendRedirect("/"+getServletContext().getServletContextName() +"/gameWindow.jsp?challengedPlayer=" + challengingPlayer + "&startedTheGame=" + startedTheGame);
}
```

Figura 12 - Servlet - AnswerChallenge – Reencaminhamento

Tal como acontece no “playerLobby.jsp”, existem dois métodos a serem chamados no “window.onload” do javascript associado. O primeiro é simplesmente obter a informação inicial. O segundo é, tal como anteriormente, um temporizador que atualiza o estado do jogo a cada segundo.

```
<body>

    <h2 id="playingVersusMessage">Playing vs </h2>
    <h2><%=request.getParameter("startedTheGame")%></h2>

    <div id="gameBoard">
        <%
        for(int col = 0; col < 8; col++)
        {
        %>
            <div class="row">
                <%
                for(int row = 0; row < 8; row++)
                {
                    %>
                    <div class="slot" id="<%= row %> <%= col %>"><p><%= row %> <%= col %></p></div>
                    <%
                }
                %>
            </div>
        <%
        }
        %>
    </div>


    <div id="gameButtons">
    </div>
</body>
```

Figura 13 - gameWindow.jsp

```
window.onload = function()
{
    setInitialInfo();
    setInterval(updateGame, 1000);
}
```

Figura 14 - Javascript - gameWindow.jsp – onload

```
function setInitialInfo()
{
    xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function(){
        if(xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            res = xmlhttp.responseText;
            var initialInfo = res.split(" ");

            var playingAgainst = document.getElementById("playingVersusMessage");
            playingAgainst.innerHTML = "Playing against " + initialInfo[0];
        }
    }
    xmlhttp.open("GET","GameWindowInitialInfo", true);
    xmlhttp.send();
}
```

Figura 15 - Javascript - gameWindow.jsp – setInitialInfo

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    response.setContentType("application/json");
    ServletOutputStream out = response.getOutputStream();

    HttpSession session = request.getSession();

    String challenged = (String) session.getAttribute("challengedPlayer");

    System.out.println("CHALLENGED: " + challenged);

    if(challenged != null)
    {
        out.print(challenged + " ");
    }

    if(session.getAttribute("startedTheGame") != null)
    {
        System.out.println("STARTED GAME: " + session.getAttribute("startedTheGame"));

        out.print((String) session.getAttribute("startedTheGame"));
    }

    out.println();
}
```

Figura 16 - Servlet – GameWindowGetInitialInfo

```javascript
if(isGameOver == "true")
{
    //TIE
    if(currentTurnPlayerName.trim() == "null")
    {
        popUpGameOver("TIE");
    }
    else
    {
        if(currentTurnPlayerName.trim() == playerName.trim())
        {
            popUpGameOver("LOST");
        }
        else
        {
            popUpGameOver("WON");
        }
    }

    updateGameBoard(gameBoard);
}
else
{//Não acabou o jogo
//Atualizar tabuleiro
//Adicionar (ou não) os botões para as jogadas.
    updateGameBoard(gameBoard);

    if(currentTurnPlayerName.trim() == playerName.trim())
    {
        addButtons(buttonColumns);
    }
}
```

Figura 17 - Javascript - gameWindow.jsp – updateGame

```
if(message == null)
{
    return;
}

String[] parameters = Message.analiseMessage(message);

if(!parameters[0].equals("UpdateGameState"))
{
    return;
}

for(int index = 0; index < parameters.length; index++)
{
    if(index == 0 || index == 1 || index == 2)
    {
        parameters[index].replace("\n", "").replace("\r", "");
    }

    if(parameters[index].equals(""))
    {
        parameters[index] = "_";
    }

    out.print(parameters[index] + " ");
}
```

Figura 18 - Servlet – GameWindowUpdateGame

Ao carregar num dos botões representantes das colunas, é chamado o método “playColumn” que, por sua vez, faz um pedido à Servlet com o mesmo nome que envia uma mensagem ao Servidor com a informação necessária.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    HttpSession session = request.getSession();

    Client client = ((Client) session.getAttribute("client"));
    String me = client.getMyInfo().getClientName().replace("\n", "").replace("\r", "");

    String column = request.getParameter("column");

    client.PlayColumn(me, Integer.parseInt(column));
}
```

Figura 19 - Servlet - PlayColumn

6. Conclusões

Com ambas as partes deste trabalho concluído, foi possível perceber o processo completo do “crossplay”. Um conceito importante nos dias de hoje dado que não é necessário possuir um cliente específico da plataforma em uso. Com ligação à internet e uso de um browser é possível efetuar as mesmas operações que com o cliente.

Foi desenvolvido conhecimento acerca dos Servidores Web, do funcionamento dos JSP’s, assim como das Servlets e da sua utilidade para expandir o servidor.

7. Notas Finais

Apesar do trabalho estar completo, existem vários locais onde são possíveis melhorias. Em particular, na parte gráfica, tanto quanto ao Cliente como no browser.

Também ficaram funcionalidades por implementar, em ambas as partes do trabalho.

Caso estas melhorias fossem de facto efetuadas, acredito que o trabalho teria melhor aspeto, seria mais completo e, por sua vez, mais apelativo.

Devido ao tempo limitado, foi crucial tomar decisões de prioridades de implementação. O que foi implementado foi considerado necessário para o projeto.

8. Bibliografia

[1] ICD, Slides, “World Wide Web”.

[2] “ICD, Slides, “WEB – Geração de Páginas Dinamicamente – Servlet’s”.

[3] “Servidor Web”, [Online]. “https://pt.wikipedia.org/wiki/Servidor_web”.